Solution Adaptive Mesh Optimization Using Algebraic Mesh Quality Metrics

Kyle Chand
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, California
www.llnl.gov/CASC/Overture

Overture team: David Brown, Kyle Chand, Petri Fast,
Bill Henshaw, Brian Miller, Anders Petersson,
Dan Quinlan, Marcus Schordan, Qing Yi

This work was performed under the auspices of the U.S. Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Introduction

ALE (Lagrange+Remap) algorithms:

- often use local mesh adjustments improve/untangle meshes
- geometry based optimization seems to be standard (eg smoothing)

Better to adapt the mesh to reduce the error in the solution:

- geometric “niceness” is often insufficient, ignores solution behavior
- the mesh spacing requirements change during the simulation
- multi-physics simulations often have competing demands
- adjust mesh to capture unresolved features (shocks, etc)

Of course, we would like:

- to adjust the mesh only in regions that require it
- minimal user interaction

Outline

Review of algebraic mesh quality metrics

Mesh optimization procedure

Error indicators

Concocting a Jacobian from error indicators

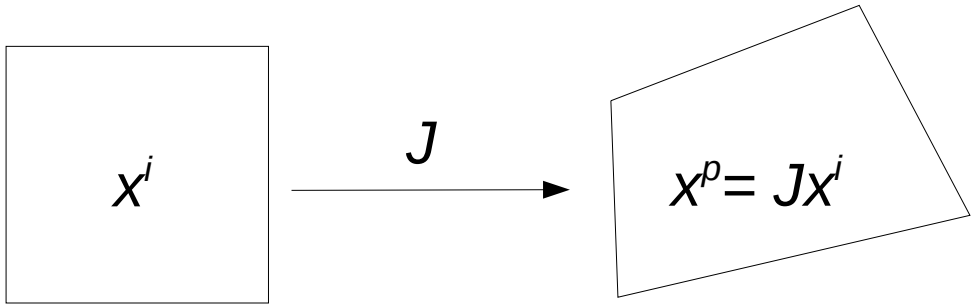
Demonstrations

Remarks

Mesh quality

Mesh quality assessment based on P. Knupp's Algebraic Mesh Quality metrics (Knupp '99).

These metrics use properties of the Jacobian of the (linear) mapping between the actual and the "ideal" element:



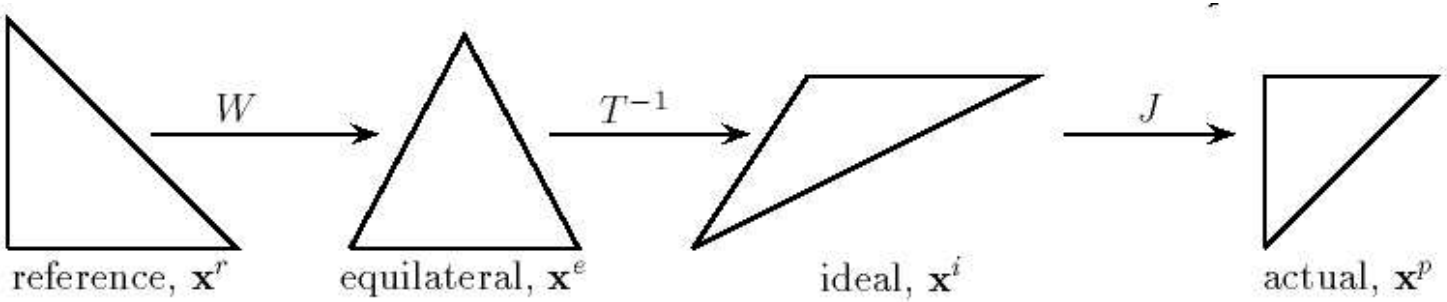
Useful metrics include :

- det(J) – scaled size
- K(J) - Condition number or C/K(J) - "shape" metric
- min(det(J), 1/det(J)) C / K(J) – combined shape and size metric

Mesh quality

Computing the Jacobian between the "ideal" element and the actual element (Pat Knupp) :

$$A = JT^{-1}W = A(\mathbf{x}^p)$$



$$J = AW^{-1}T = AM$$

W : determined by the shape of the element
T : derived (later) based on error estimates
A : from the actual element vertices

Problem: Derive T to reduce discretization errors

Element Jacobian calculation

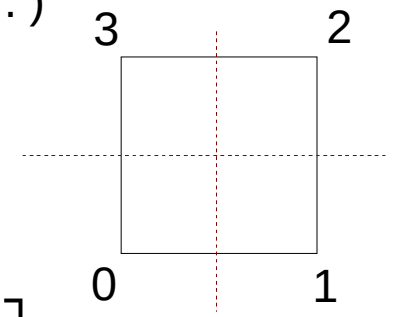
$$A^{tri} = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix}$$

Same as P. Knupp for triangles and tets

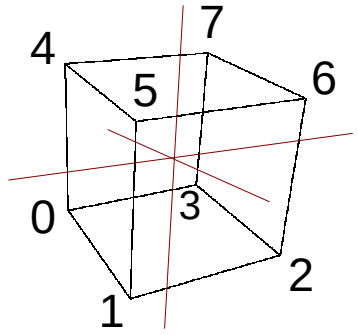
$$A^{tet} = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{bmatrix}$$

Centered finite difference to compute the derivatives for quads and hexes:
(note that det(J)>0 for "slightly" tangled quads and hexes...)

$$A^{quad} = \frac{1}{2} \begin{bmatrix} x_1 + x_2 - x_0 - x_3 & x_2 + x_3 - x_0 - x_1 \\ y_1 + y_2 - y_0 - y_3 & y_2 + y_3 - y_0 - y_1 \end{bmatrix}$$



$$A^{hex} = \frac{1}{4} \begin{bmatrix} x_{2367} - x_{0154} & x_{0374} - x_{1265} & x_{4567} - x_{0123} \\ y_{2367} - y_{0154} & y_{0374} - y_{1265} & y_{4567} - y_{0123} \\ z_{2367} - z_{0154} & z_{0374} - z_{1265} & z_{4567} - z_{0123} \end{bmatrix}$$



Use the quad/hex corner to eliminate +/- (bowtie) oscillations in the mesh

Mesh optimization

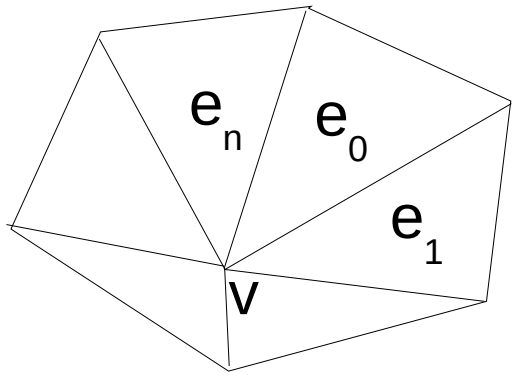
Local mesh improvement based on nonlinear optimization of vertex locations (Lori Frietag, Pat Knupp '99, '00, ...)

Define : $f_v = f(x_v) = f(J_0(x_v), J_1(x_v) , ..., J_n(x_v))$

= the objective function at vertex v ($J_e = A_e M_e$)

$$f_v(\mathbf{x}_v) = \sum_{e=0}^{n} f_e(J_e(\mathbf{x}_v))$$

$$= \sum \kappa_e^2 \qquad \textit{(for example)}$$



Candidate element objective functions include:

- condition number (above)
- determinant (max(det(J), 1/det(J)))
- ...

Mesh optimization

Analytic derivatives of the objective function can be computed since M is a constant in each element and A is a function of x_v:

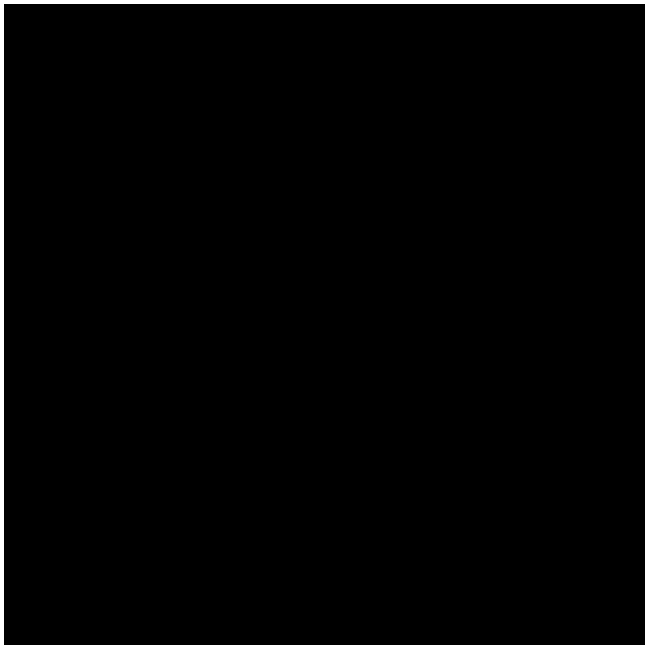
$$\frac{\partial f_e(J)}{\partial x_v} = tr\left(\frac{\partial f_e}{\partial J}\frac{\partial J}{\partial x_v}^T\right) = tr\left(\frac{\partial f_e}{\partial J}\left(\frac{\partial A}{\partial x_v}M\right)^T\right)$$

Steepest descent can be used to optimize the vertex position:

$$\mathbf{d} = -d\nabla f_v \longrightarrow \mathbf{x}_v^{n+1} = \mathbf{x}_v^n + \mathbf{d}$$

Iteratively search for an optimal step size using a quadratic line search

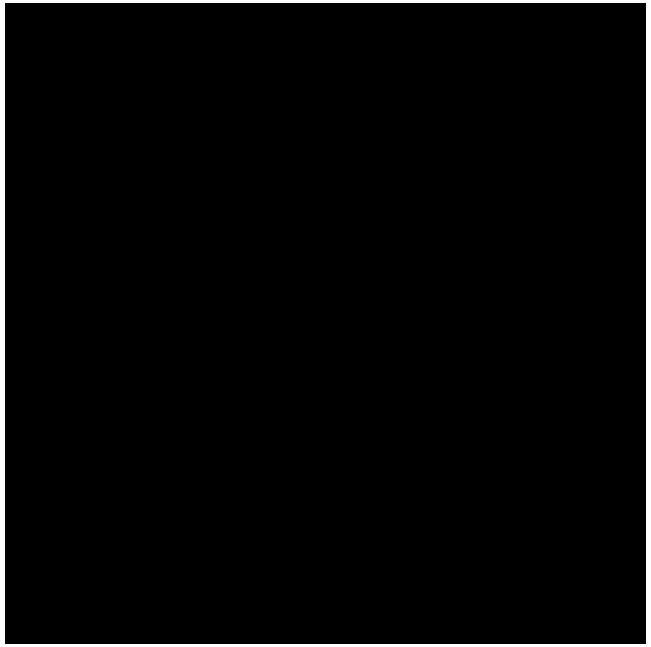
Mesh optimization, 2D results



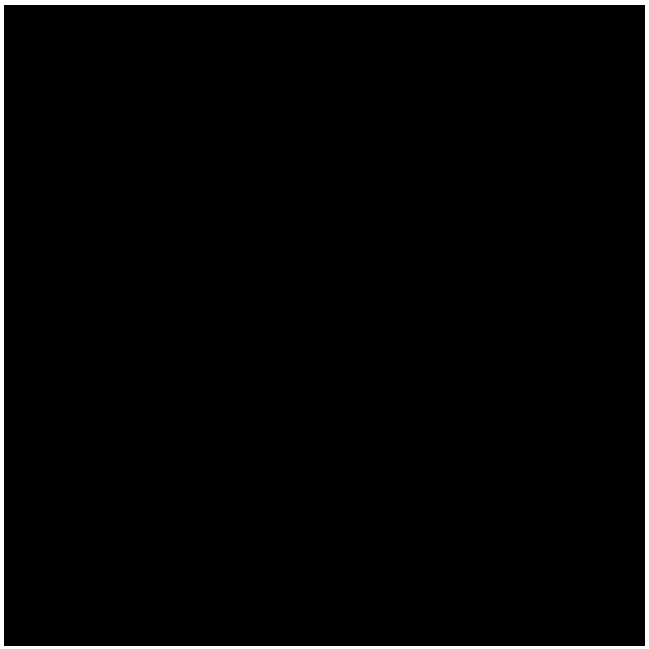
3 optimization sweeps

M = *I*

ideal is a uniform quad



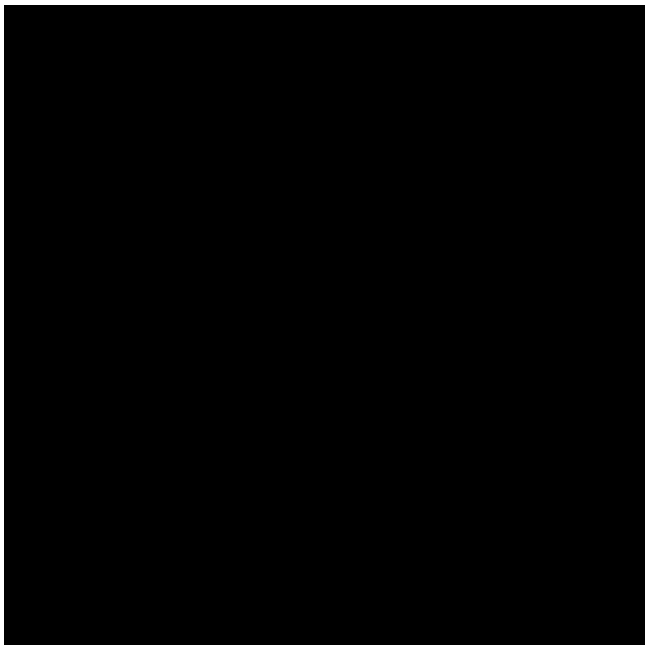
2D Triangle optimization

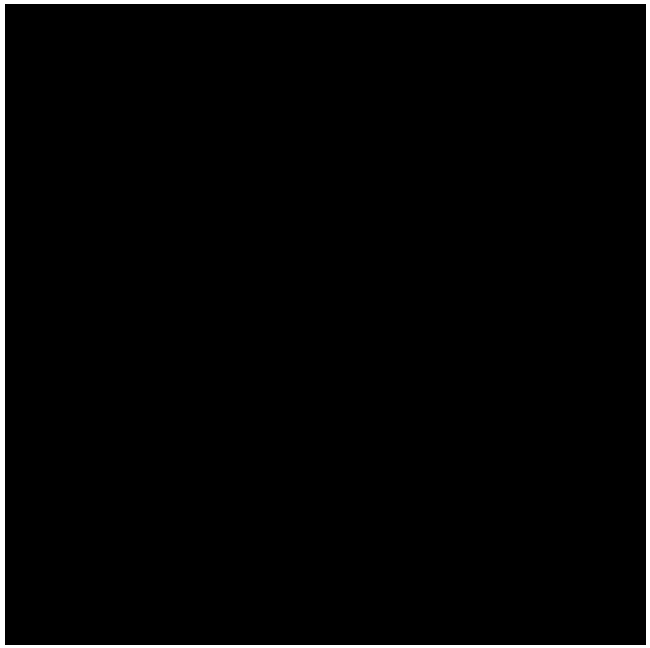


3 optimization sweeps

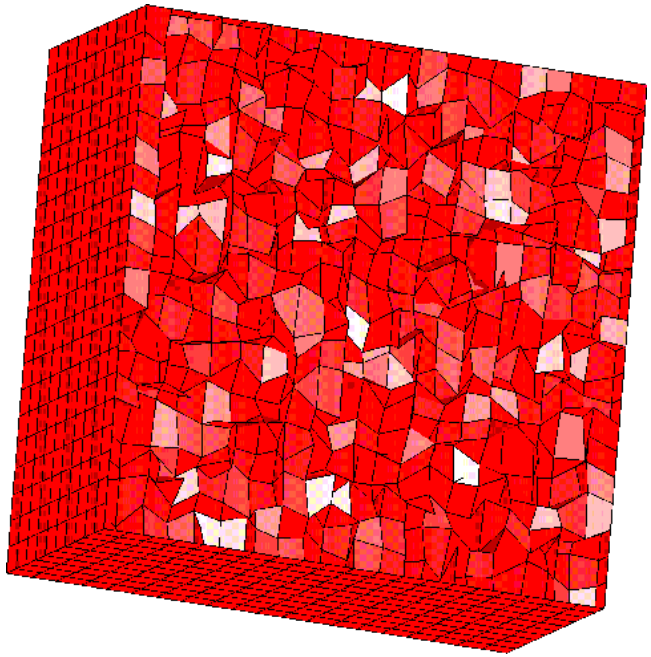
M = *I*

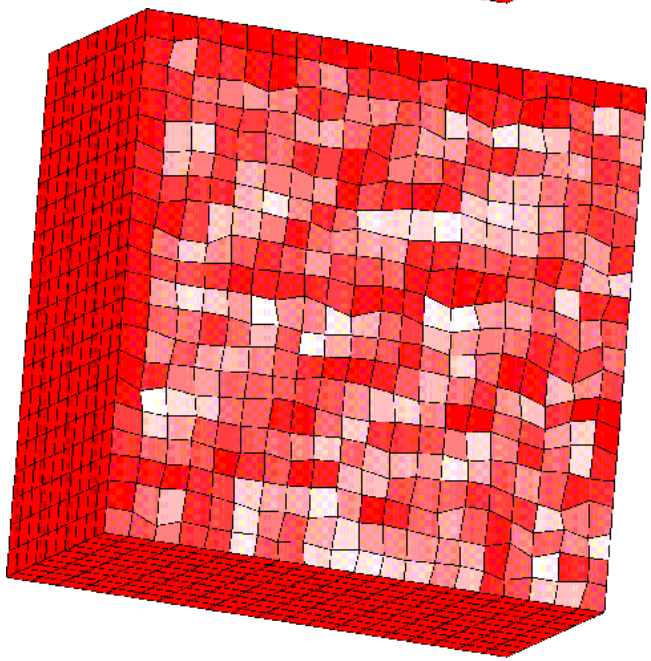
ideal is a right triangle





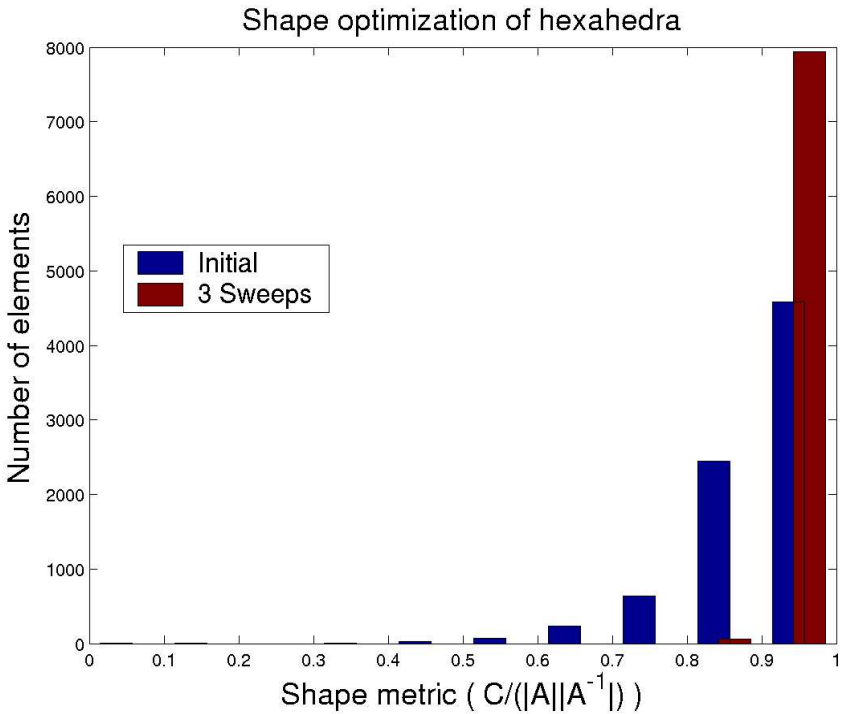
3D Examples





3 optimization sweeps

M = *I*
(ideal is a uniform hex)



Solution adaptive optimization

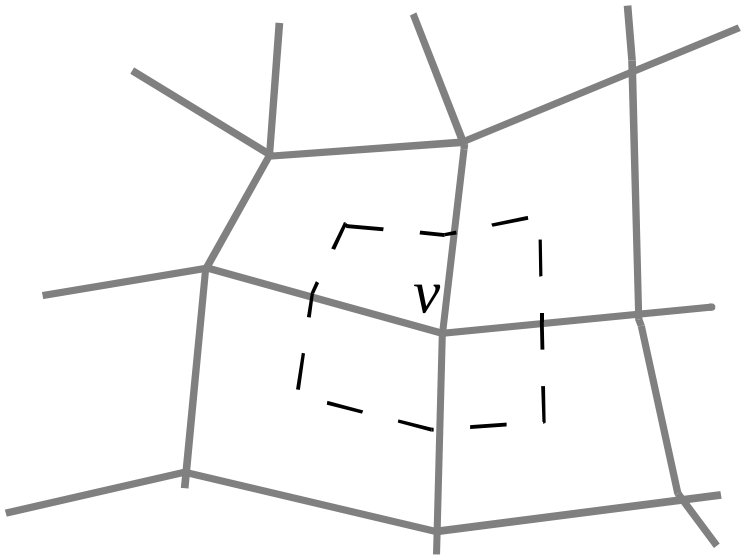
$$J = AW^{-1}T = AM$$

Problem: Derive T to reduce discretization errors

T will be constructed to scale the element size down in regions where the error is (probably) large.

The objective function should incorporate both size and shape.

Error Indicators



$$e^v = \sum_{i=1}^{nDim} \alpha_i \left|\Delta_{1i}^v\right| + \beta \left|\Delta_2^v\right|$$

$$\eta = \frac{e}{max(e)}$$ (normalized indicator)

Use a generalization of undivided differences

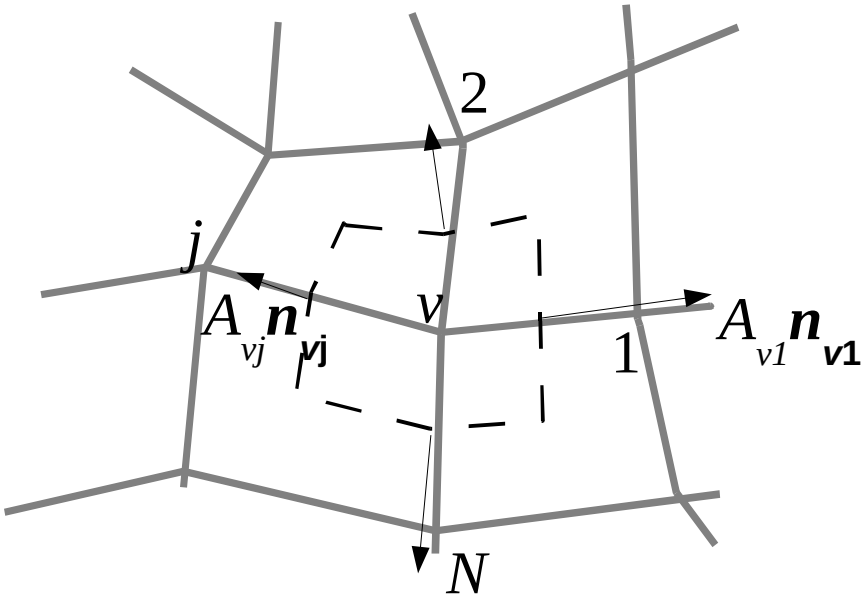
1st Order undivided difference Δ_{1i}^v :

finite volume difference divided by circumference (surface area) instead of volume

2nd Order undivided difference Δ_2^v :

average of the neighbors – vertex value

Error Indicators (details)



$\boldsymbol{n}_{vj}$ = unit vector for face normal

A_{vj} = area for face between *v* and *j*

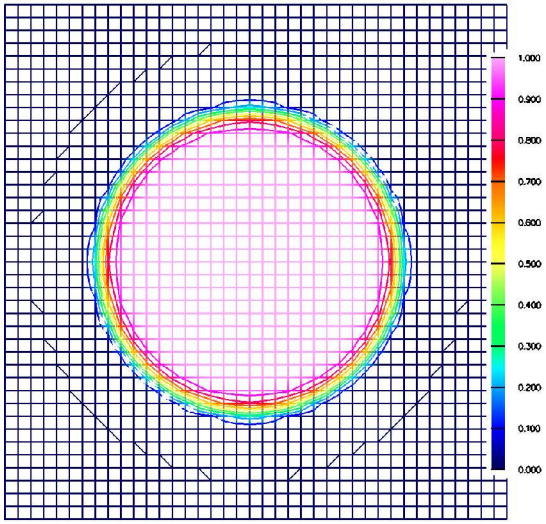
$u_{vj} = .5\,(u_v + u_j)$

N = number of adjacent vertices

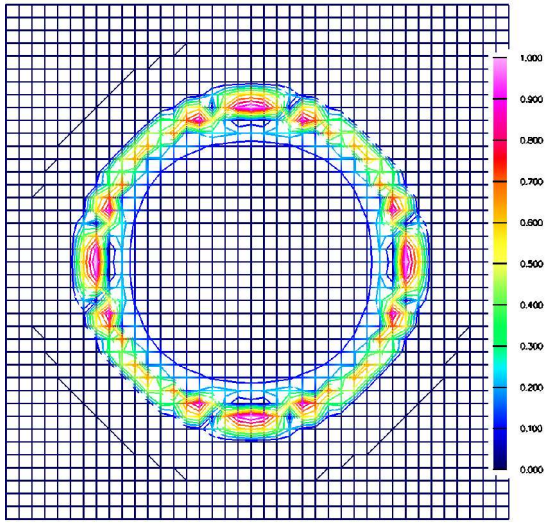
$$\Delta_{1i}^{v} = \frac{\sum_j u_{vj} A_{vj} n_{vj}^{i}}{\sum_j A_{vj}}$$

$$\Delta_{2}^{v} = \frac{1}{N}\sum_j u_j - u_v$$

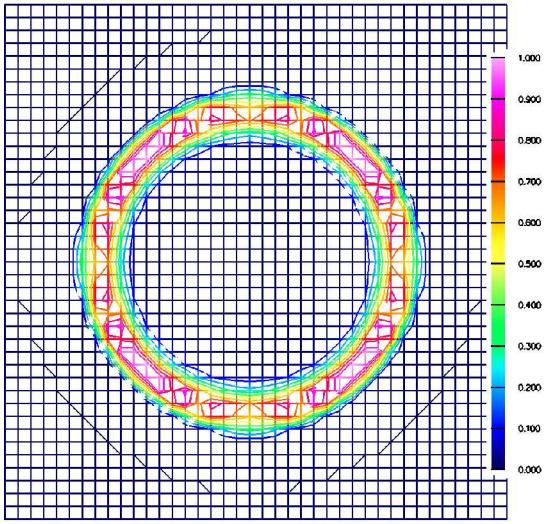
Error Indicators



function



normalized gradient error



error indicator

Robust and efficient indicator of discretization error

η --> 0 as the grid is refined and the function is resolved

Easy to implement on general unstructured grids

“Ideal” Jacobian from Error Indicators

Use the error indicator to scale cell volumes:

shrink cells where the error appears high

leave them alone if the indicator is close to zero

1. Smooth η using 5-10 Jacobi iterations (depending on grid size and error indicator variation)
2. Concoct T to scale the current cell volumes down
   - η lives at the vertices:
     average them over the vertices in the cell
     or take the maximum value of the cell vertices
   - scale the volume to get an isotropic reduction in cell size

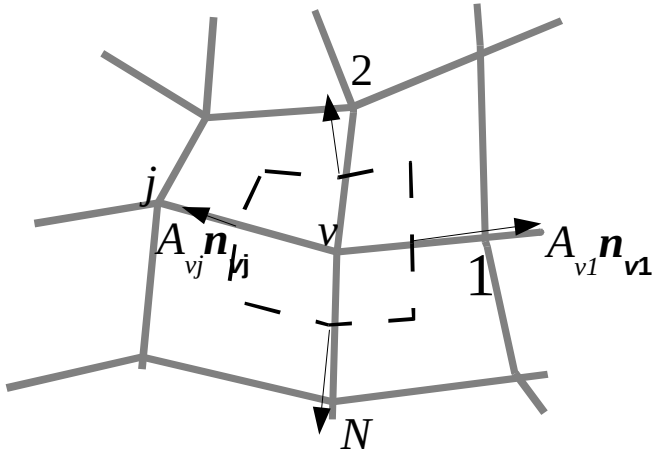
$$f = (1-\eta)^p$$ *0<p<4* works well

$$V_{new} = max(fV_{old}, V_{min})$$

$$T = V_{new}^{-\frac{1}{d}} I$$

Details:
discrete gradient, objective function

Node centered finite volume (2nd order accurate):



$\mathbf{n}_{vj}$ = unit vector for face normal

A_{vj} = area for face between *v* and *j*

$u_{vj} = .5\,(u_v + u_j)$

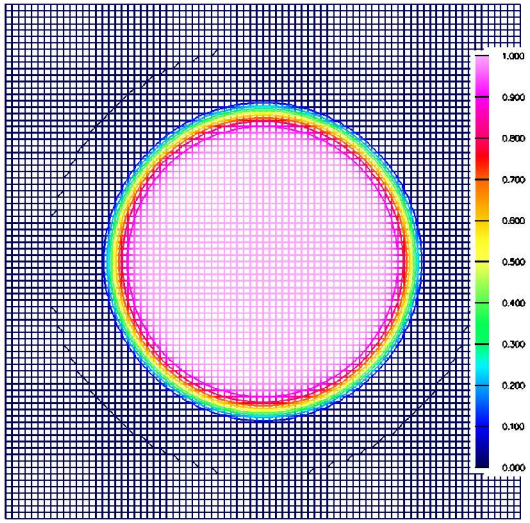
N = number of adjacent vertices

$$\nabla u\,|_v \approx \frac{\sum_j u_{vj} A_{vj} \mathbf{n}_{vj}}{V_v}$$

Objective function for the size and shape:

$$f_e = \kappa^2 + max\left(det\,|J|\,, \frac{1}{det\,|J|}\right)$$

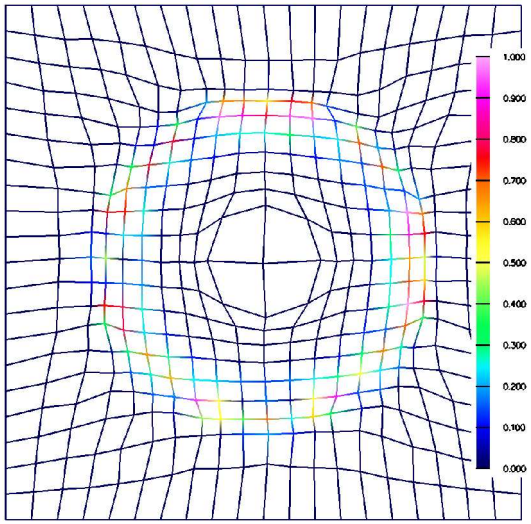
2D Example, non-moving mesh



$u = \exp(5e5|\mathbf{x}-\mathbf{x}_0|^{20})$

5 optimization sweeps through the mesh

Note that the pulse is not well resolved



Initial:

$L_\infty = 13.41$

$L_1 = 1.398$

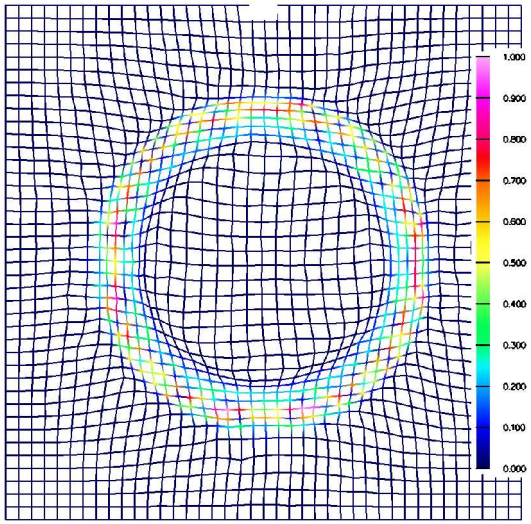
$L_2 = 3.250$

Optimized:

$L_\infty = 9.378$

$L_1 = 0.8151$

$L_2 = 2.021$



Initial:

$L_\infty = 7.412$

$L_1 = 0.5077$

$L_2 = 1.398$

Optimized:

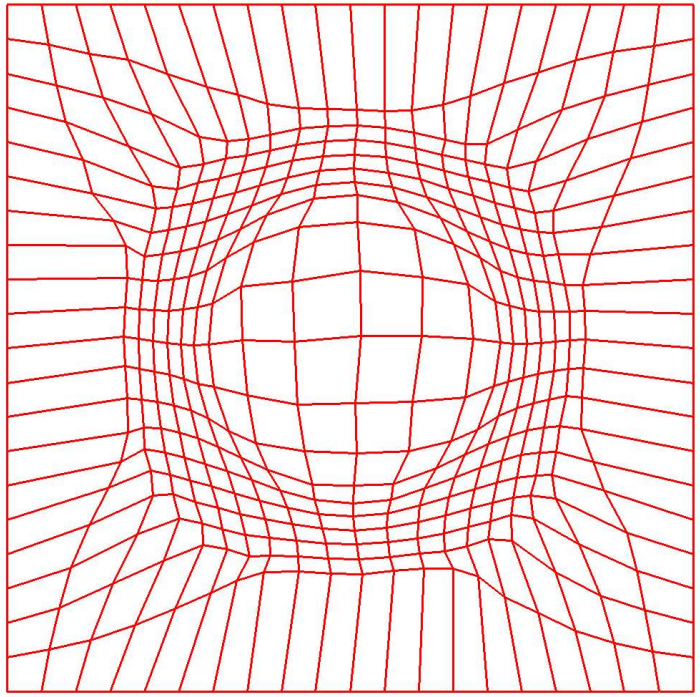
$L_\infty = 4.301$

$L_1 = 0.2439$

$L_2 = 0.6966$

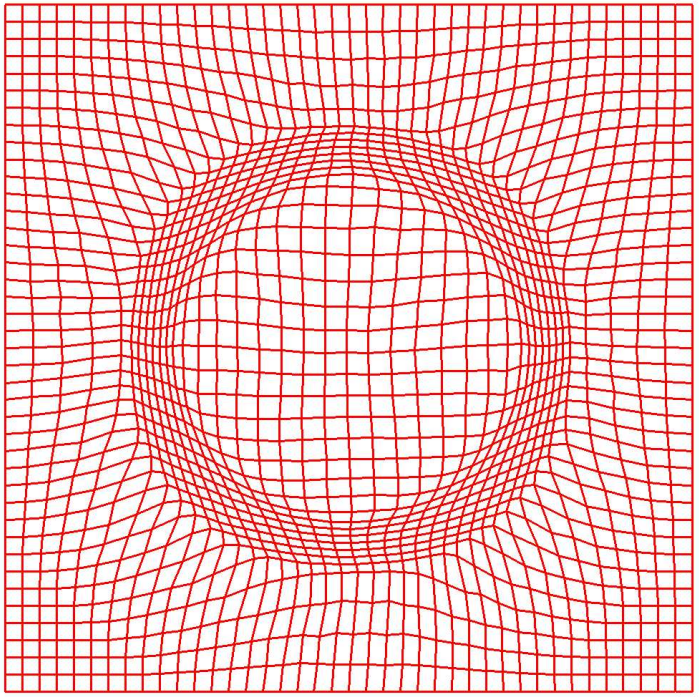
2D Example, non-moving mesh

Same problem, iterate 6 times:
evaluate the test function
perform 3 optimization sweeps



Initial:
$L_\infty = 13.41$
$L_1 = 1.398$
$L_2 = 3.250$

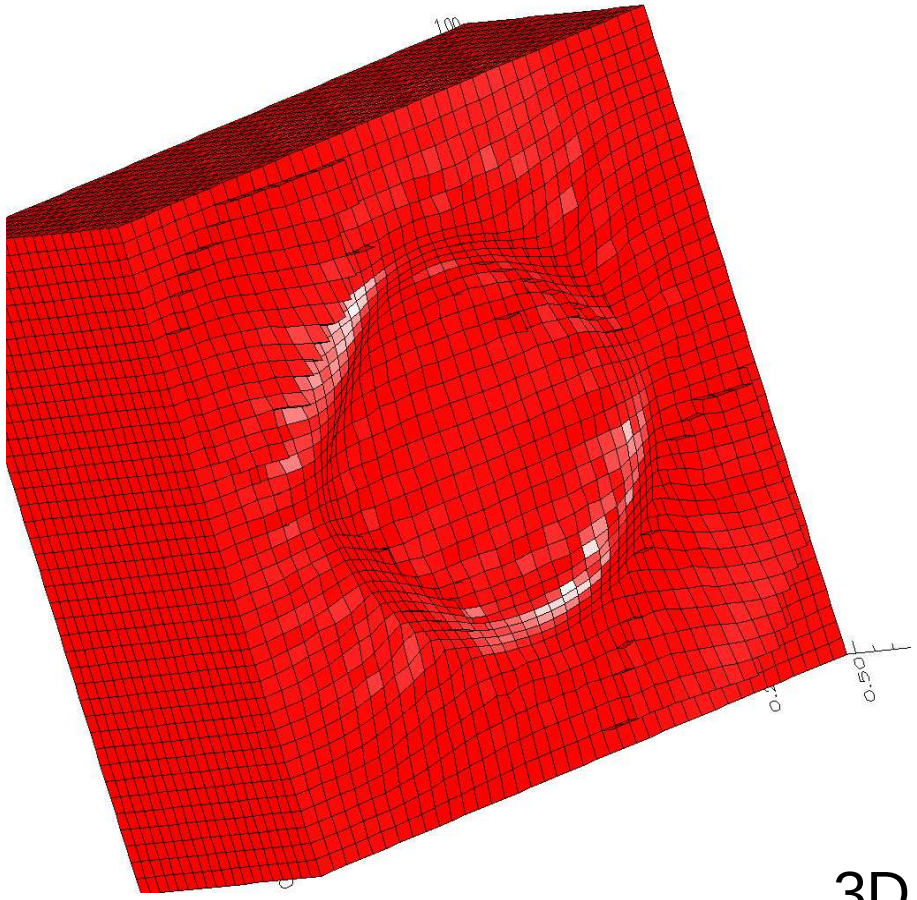
Optimized:
$L_\infty = 6.656$
$L_1 = 0.5129$
$L_2 = 1.284$



Initial:
$L_\infty = 7.412$
$L_1 = 0.5077$
$L_2 = 1.398$

Optimized:
$L_\infty = 2.110$
$L_1 = 0.1040$
$L_2 = 0.3090$

3D Example, non-moving mesh



Initial:

$L_\infty = 7.412$

$L_1 = 0.2608$

$L_2 = 0.9418$

Optimized:

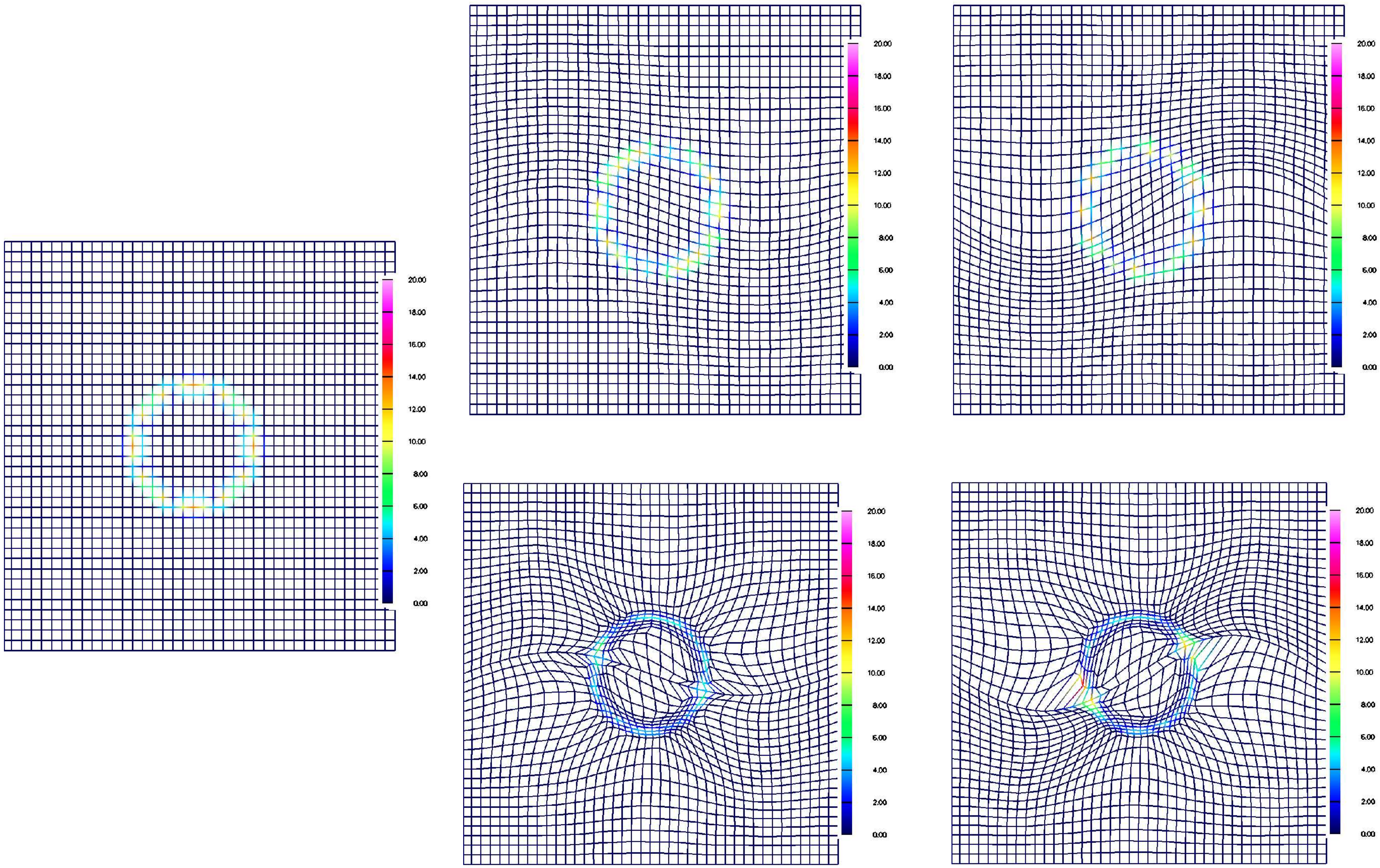
$L_\infty = 8.159$

$L_1 = 0.1324$

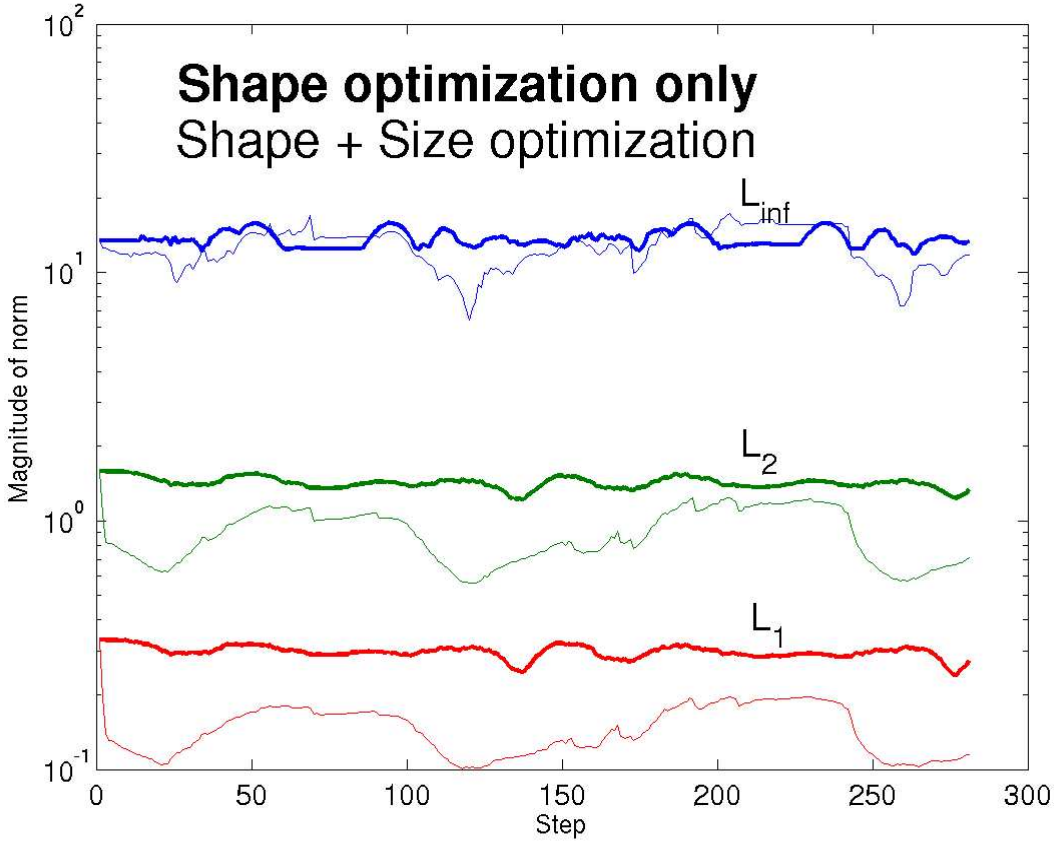
$L_2 = 0.5660$

3D is not reliable yet

2D Example, mesh with forced motion



2D Example, mesh with forced motion



L_1 and L_2 norms improve with size optimization

Inf. norm stays the same, nodes constrained to the interface have the highest error

Remarks

Remap codes could benefit from remeshing algorithms that take discretization error into account.

The presented method is straightforward to implement in codes with similar optimization mechanisms. New software (Mesquite) will make adding optimization to existing codes easier.

Future work:

- more sophisticated scaling method
- actually try this in an ALE code
- define objective functions based on multiple solution components
- anisotropic optimization